PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06K 9/62 | A1

(11) International Publication Number: WO 93/18484

(43) International Publication Date: 16 September 1993 (16.09.93)

(21) International Application Number: PCT/US93/02179

(22) International Filing Date: 10 March 1993 (10.03.93)

(30) Priority data: 07/848,833 10 March 1992 (10.03.92) US

(71) Applicant: ORACLE CORPORATION [US/US]; 500 Oracle Parkway, Redwood Shores, CA 94065 (US).

(72) Inventor: YU, Francis, H. ; 2001 Southwest 15th Street, Apartment A22, Gainesville, FL 32608 (US).

(74) Agent: HARRIMAN, J., D., II; Hecker & Harriman, 2049 Century Park East, Suite 1200, Los Angeles, CA 90067 (US).

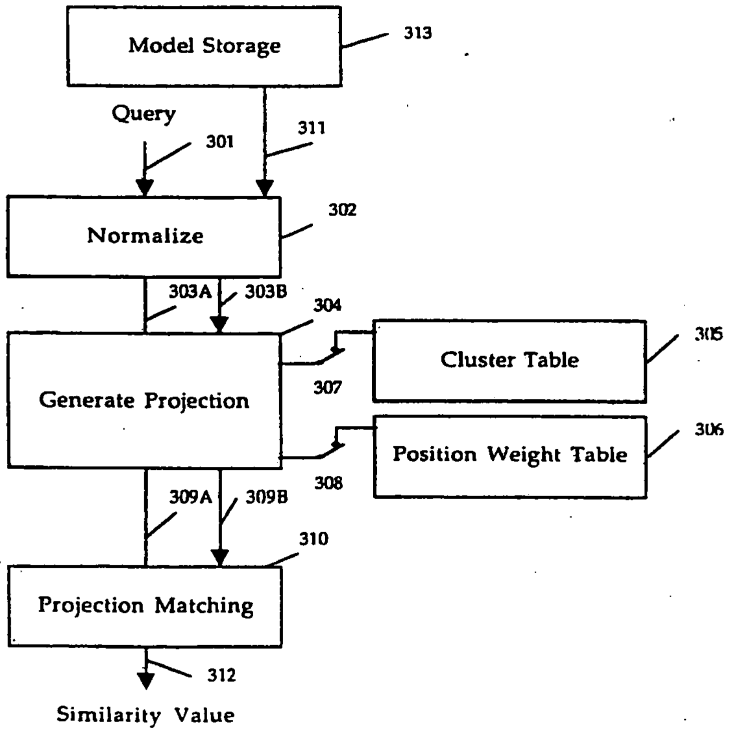
(81) Designated States: AT, AU, BB, BG, BR, CA, CH, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, LK, LU, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SK, UA, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR COMPARISON OF DATA STRINGS

(57) Abstract

The present invention is a method and apparatus that measures the similarity of two images. Any information that can be discretely symbolized can be transformed into an image through so-called "image projection". This process is used to define otherwise discrete entities as part of a linear space, making it possible to calculate distances among those entities. A mechanism called a cluster allows association of otherwise discrete symbols, improving the matching abilities of the invention. Initially, the sequence of symbols is normalized (302). Then a projection (304) of the normalized sequence is created. The projection may be optionally generated with a cluster (305) that assigns weights to the neighbors of a core symbol and/or with position weights (306) that assigns weights to each position in the normalized image. Projection matching (310) is then performed to determine match candidate for the string of symbols.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	MR	Mauritania
AU	Australia	GA	Gabon	MW	Malawi
BB	Barbados	GB	United Kingdom	NL	Netherlands
BE	Belgium	GN	Guinea	NO	Norway
BF	Burkina Faso	GR	Greece	NZ	New Zealand
BG	Bulgaria	HU	Hungary	PL	Poland
BJ	Benin	IE	Ireland	PT	Portugal
BR	Brazil	IT	Italy	RO	Romania
CA	Canada	JP	Japan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SK	Slovak Republic
CI	Côte d'Ivoire	KZ	Kazakhstan	SN	Senegal
CM	Cameroon	LI	Liechtenstein	SU	Soviet Union
CS	Czechoslovakia	LK	Sri Lanka	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	MC	Monaco	UA	Ukraine
DK	Denmark	MG	Madagascar	US	United States of America
ES	Spain	ML	Mali	VN	Viet Nam
FI	Finland	MN	Mongolia		

METHOD AND APPARATUS FOR COMPARISON OF DATA STRINGS

FIELD OF THE INVENTION

This invention relates to the field of data comparison.

BACKGROUND ART

In data storage systems or data base systems, it is often desired to retrieve blocks of data in response to a query. In other cases, an unknown block of data is compared with stored blocks of data as a means of identifying the unknown block of data. In some cases, there is no stored block of data in the data base that matches the query. Similarly, there may be no matching stored block of data for a given unknown block of data. However, it may be useful to provide information about the blocks of data that are closest to matching the query block of data. This is particularly true in spell check programs where a word is misspelled and the most likely replacement word is to be determined. A system for determining the best match for a particular block of data is known as a word comparator, string matching scheme, or matching algorithm.

In the prior art, such matching is accomplished by relatively straightforward algorithms that seek to identify common characters or symbols between two strings. For example, a "left-to-right" comparison of two strings is performed until common characters are found. The common characters are then aligned and a "right-to-left" comparison is performed. This algorithm only identifies typographic differences between two strings.

There are prior art patents that describe matching schemes that include methods for determining the degree of similarity between two strings. Both Parvin 4,698,751 and Parvin 4,845,610 describe a string to string matching method in which a "distance" between the two strings is calculated. "Distance" in Parvin is defined as the minimum number of editing operations (such as adding a character, deleting a character and substituting for a character) needed to convert one string to the other.

Yu et al., U. S. Patent 4,760,523, describes a "fast search processor" for searching for a predetermined pattern of characters. The processor includes serially connected cells each of which contain a portion of the pattern. The character set being searched is sent in a serial fashion through the serially connected cells. Match indicators record each match between the pattern in a cell and the character stream flowing through the cell.

Hartzband et al., U. S. Patent 4,905,162 describes a method for determining the similarity between objects having characteristics that are specified on a reference table. Weights for each characteristic may be specified by a user. A numerical value for the similarity between objects is calculated based on an element by element comparison of each characteristic.

U. S. Patent 4,979,227 to Mittelbach et al. describes a method, in an optical character recognition context, for recognizing a character string by comparing the string to a lexicon of acceptable character strings. The best matching character strings from the lexicon are selected, and tested to see whether substitutions that would convert the original string to the lexicon string are permitted. An example of a permitted substitution would be substituting an "l" for an "i", since these characters are similar in appearance. The actual comparison process is not described in this patent.

Fujisawa et al., U. S. Patent 4,985,863 describes a document storage and retrieval system in which both image and text files of the document are stored in memory. The desired image file is selected by searching the associated text file. The text file, which may be generated by optical character recognition methods applied to the image files, contains special characters that indicate ambiguous characters. Possible alternatives may be provided for an ambiguous character. For example, if a character is recognized as being possibly an "o" or an "a", both these characters are listed together with the special characters indicating the existence of an ambiguity.

U. S. Patent 5,008,818 to Bocast describes a method and apparatus for reconstructing altered data strings by comparing an unreconstructed string to "vocabulary" strings. The comparison is done on a character by character basis by moving pointers from the beginning to the end of the unconstructed string, one of the pointers indicating the character being compared, the second acting as a counter for the number of correct comparisons. The comparison is under certain conditions also done from the back to the front of the string. A "reconstruction index" indicating the similarity between the unconstructed string and the vocabulary string is calculated from the positions of the pointers.

U. S. Patent 5,060,143 to Lee describes a method for comparing strings of characters by comparing a target string to sequential blocks of candidate strings. By comparing the target string to sequential portions of the candidate strings, rather than to the candidate string as a whole, performance is improved by eliminating redundant comparisons. An early "time out" feature determines early during the comparison process whether the candidate string can possibly be a valid match. If not, the comparison to that candidate string is aborted and a comparison to the first block of the next candidate string is begun.

SUMMARY OF THE PRESENT INVENTION

The present invention is a method and apparatus that measures the similarity of two images. Any information that can be discretely symbolized can be transformed into an image through so-called "image projection". This process is used to define otherwise discrete entities as part of a linear space, making it possible to calculate distances among those entities. A mechanism called a cluster allows association of otherwise discrete symbols, improving the matching abilities of the invention. Cluster tables are created that reflect symbol relationships. By adjusting the cluster tables, the outcome of similarity ranking can be controlled.

The invention is used to measure the similarity between two strings of symbols. The invention generates scaled scores that represent the degree of matching between two vectors. The invention can be used as a spelling correction tool, a phonetic matching scheme, etc.

The process of image projection transforms a string into a real-valued vector. When searching for best matches in a large space, projection vectors can be used to create an index in the search space. With a proper indexing method, the best matches for a query can be found in the same time as required to search for an exact match.

The present invention operates in several steps. Initially, the sequence of symbols is normalized. Then, a projection of the normalized sequence is created. The projection may optionally be generated with a cluster that assigns weights to the neighbors of a core symbol and/or with position weights that assigns weights to each position in the normalized image. Projection matching is then performed to determine match candidates for the string of symbols.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a flow diagram of the operation of the present invention.

Figure 2 is a flow diagram illustrating the preferred embodiment of the present invention.

Figure 3 is a block diagram illustrating the preferred embodiment of the present invention.

Figure 4 is a block diagram of an example of a computer system for implementing the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A method and apparatus for comparing data strings is described. In the following description, numerous specific details, such as normalized, normalization values, weight values, etc., are described in order to provide a more thorough description of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances well known features have not been described in detail so as not to obscure the present invention.

The present invention operates as follows:

1. Normalize sequence of symbols.
2. Create projection of normalized sequence. (Optionally with a cluster that assigns weights to the neighbors of a core symbol and/or with position weights that assign weights to each position in the normalized image.)
3. Perform projection matching.

The present invention may be implemented on any conventional or general purpose computer system. An example of one embodiment of a computer system for implementing this invention is illustrated in Figure 4. A keyboard 410 and mouse 411 are coupled to a bi-directional system bus 419. The keyboard and mouse are for introducing user input to the computer system and communicating that user input to CPU 413. The computer system of Figure 4 also includes a video memory 414, main memory 415 and mass storage 412, all coupled to bi-directional system bus 419 along with keyboard 410, mouse 411 and CPU 413. The mass storage 412 may include both fixed and removable media, such as magnetic, optical or magnetic optical storage systems or any other available mass storage technology. The mass storage may be shared on a network, or it may be dedicated mass storage. Bus 419 may contain, for example, 32 address lines for addressing video memory 414 or main memory 415. The system bus 419 also includes, for example, a 32-bit data bus for transferring data between and among the components, such as CPU 413, main memory 415, video memory 414 and mass storage 412. Alternatively, multiplex data/address lines may be used instead of separate data and address lines.

In the preferred embodiment of this invention, the CPU 413 is a 32-bit microprocessor manufactured by Motorola, such as the 68030 or 68040, or Intel, such as the 80386 or 80486. However, any other suitable microprocessor or microcomputer may be utilized.

Main memory 415 is comprised of dynamic random access memory (DRAM) and in the preferred embodiment of this invention, comprises 8 megabytes of memory. More or less memory may be used without departing from the scope of this invention. Video memory 414 is a dual-ported video random access memory, and this invention consists, for example, of 256 kbytes of memory. However, more or less video memory may be provided as well.

One port of the video memory 414 is coupled to video multiplexer and shifter 416, which in turn is coupled to video amplifier 417. The video amplifier 417 is used to drive the cathode ray tube (CRT) raster monitor 418. Video multiplexing shifter circuitry 416 and video amplifier 417 are well known in the art and may be implemented by any suitable means. This circuitry converts pixel data stored in video memory 414 to a raster signal suitable for use by monitor 418. Monitor 418 is a type of monitor suitable for displaying graphic images, and in the preferred embodiment of this invention, has a resolution of approximately 1020 x 832. Other resolution monitors may be utilized in this invention.

The computer system described above is for purposes of example only. The present invention may be implemented in any type of computer system or programming or processing environment.

A flow diagram illustrating the operation of the present invention is illustrated in Figure 1. At step 101, the symbol sequence to be compared is identified and prepared. This involves normalizing the sequence. At step 102, a projection of the normalized sequence is generated. The projection can be generated with one or both of cluster table 103 and weight table 104.

At step 105, the output of step 102, a real valued vector projection, is compared to other vector projections in a projection matching step.

NORMALIZATION

A string of S symbols is stretched or compressed into a normalized image of N symbols. The size of each symbol in the normalized image represents its portion in the string. Suppose the string of symbols is the word "lists", consisting of five letters or symbols ($|S| = 5$) as shown below:

l	i	s	t	s

Consider the case where the normalized number of symbols is eight so that N = 8. The medium of a symbol, M, in normalized image is computed as follows:

$$M(S_i) = i * |N| / |S|$$

where S_i is the i-th symbol in string S, $|N|$ is the normalized size, and $|S|$ is the length of string S. The five symbols of the word list are now compressed into eight symbols, with the medium of each symbol being 1.6i, $((N/S) = (8/5) = 1.6)$. The normalized size of each symbol is therefor 1.6 normal symbol slots.

Each symbol in the normalized string must have a unitary value. Therefor "l" is placed in the first symbol slot, leaving 0.6l to be placed in the second symbol slot, as shown below. To provide a unitary value for the second symbol slot, 0.4i is added to 0.6l. This leaves 1.2i. 1.0i or "i" is placed in the third symbol slot, leaving 0.2i for the fourth symbol slot and so on as shown below. In summary, each symbol from the original string is represented by 1.6 times that symbol in the normalized string.

l	0.6l+0.4i	i	0.2i+0.8s	0.8s+0.2t	t	0.4t+0.6s	S

SYMBOL PROJECTION

A projection is a real-valued vector that is equally divided into as many partitions as members in a symbol set. For example, the symbol set for a spelling checker is the set of symbols of the alphabet, numeric characters, and punctuation marks. Each partition is called a "closure" C_i for its corresponding symbol i. (A closure is larger than a normalized image).

Each symbol in the image is projected onto its closure in normal distribution, with the maximum at its medium. A decreasing series D with length |D| can be defined to simulate the normal distribution. D is called distributing series, and |D| distribution size. The projection is computed as follows:

$$c_{s_i, M(Si)+|D|+j} = d_j$$

$$c_{s_i, M(Si)+|D|-j} = d_j \qquad (j = 0, 1, 2, \ldots, |D|)$$

where d_j is j-th item in distribution series D, and C_{ik} is the k-th item in symbol S_i's closure. If a symbol occurs more than once and its distribution overlaps, only the larger values are kept.

For example, with distribution series (4, 3, 1) whose length |D| is 3, the closures for symbols L, I, S and T have a size of 12 (|N| + 2 * |D| - 2 = (8+(2*3)-2) = 12) and are as follows:

"L"	1	3	4	3	1							
"I"				1	3	4	3	1				
"S"						1	3	4	3	4	3	1
"T"						1	3	4	3	1		

Note that because there are two instances of the letter "s" in the sequence, there are two peaks. Each peak corresponds to an occurrence of "s" in the normalized stream.

The preferred embodiment of the present invention utilizes one of the following two distribution tables:

distribution table #1: {21, 19, 17, 14, 10, 4, };
distribution table #2: {17, 14, 10, 4, };

The preferred embodiment of the present invention uses the following encoding and decoding tables.

```
Encoding Table: {
    31, 31, 31, 31, 31, 31, 31, 31,                            /* 0-7 */
    31, 31, 31, 31, 31, 31, 31, 31,                            /* 8-15 */
    31, 31, 31, 31, 31, 31, 31, 31,                            /* 16-23 */
    31, 31, 31, 31, 31, 31, 31, 31,                            /* 24-31 */
    31, 31, 31, 31, 31, 31, 31, 26,                            /* 32-39 */
    31, 31, 31, 31, 31, 27, 28, 31,                            /* 40-47 */
    31, 31, 31, 31, 31, 31, 31, 31,                            /* 48-55 */
    31, 31, 31, 31, 31, 31, 31, 31,                            /* 56-63 */
    31,  0,  1,  2,  3,  4,  5,  6,                            /* 64-71 */
     7,  8,  9, 10, 11, 12, 13, 14,                            /* 72-79 */
    15, 16, 17, 18, 19, 20, 21, 22,                            /* 80-87 */
    23, 24, 25, 31, 31, 31, 31, 31,                            /* 88-95 */
    31,  0,  1,  2,  3,  4,  5,  6,                            /* 96-103 */
     7,  8,  9, 10, 11, 12, 13, 14,                            /* 104-111 */
    15, 16, 17, 18, 19, 20, 21, 22,                            /* 112-119 */
    23, 24, 25, 31, 31, 31, 31, 31     ,                       /* 120-127 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 128-143 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 144-159 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 160-175 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 176-191 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 192-207 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 208-223 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 224-239 */
    31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,           /* 240-255 */
};

Decoding Table: {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', '                       /* 0-7 */
    'i', 'j', 'k', 'l', 'm', 'n', 'o', '                       /* 8-15 */
    'q', 'r', 's', 't', 'u', 'v', 'w', '                       /* 16-23 */
    'y', 'z', '\'','-', '.', 255, 255, 255                     /* 24-31 */
};
```

PROJECTION WITH CLUSTERS

A cluster is a subset of the character set which contains a core character and any number of neighbor characters. It is used to represent relationships among characters. For example, the following cluster:

{ { MAP('a'), 8 }, { MAP('s'), 2 }, { MAP('e'), 1 }, { 0, 0 } }
indicates that 's' and 'e' are close to 'a' and 's' is closer to 'a' than 'e' is. The parameter clusters are an array of clusters, each cluster corresponds to a character in the character set as the core. Note that MAP() used above is to indicate that ASCII is usually not used for the comparison scheme of the present invention, and ASCII characters are mapped into a smaller set. The mapping function is used to reduce the size of the character set for memory optimization. The memory space for the full ASCII set may not be available. In addition, the actual symbols of interest, such as in a spelling checker, may be fewer than in the entire ASCII set. Therefore, the characters can be mapped into a smaller set. In one embodiment, characters are mapped into a space from 0 to 32.

A cluster U_i defines weights for neighbors of the core symbol i; u_{ii} is the weight of i itself. Every symbol is the core of its cluster. In simple projection, a cluster has a core as its only symbol, and the weight for the core is 1.

When a cluster has more than one symbol, or the core symbol has neighbors, the core symbol is not only projected to its own closure but also its neighbors' closures. The projection becomes:

$$c_{n\,M(s_i)+|D|+j} = d_j{}^{*}\ u_{s_i n}$$

$$c_{n\,M(s_i)+|D|-j} = d_j{}^{*}\ u_{s_i n} \qquad (j = 0, 1, 2, \ldots, |D|)$$

where n is a member of the cluster of S.

Clusters are used to associate otherwise discrete symbols. The use of clusters can provide a means to tailor queries to provide desired results. For example, consider the word "communicate" and two misspellings of that word, namely "communikate" and "communigate". It may be desirable to implement the present invention so that "communikate" shows a higher degree of matching than "communigate" (because the "k" sound is more like the hard "c" sound of "communicate"). By including "k" in the cluster of "c", the present invention can show that "communikate" is more likely to be "communicate" than is "communigate".

The present invention implements clusters through cluster tables. An example of a cluster table for the 26 lower case alpha characters is illustrated below. Each letter is followed by paired values. The first value of each pair represents the numerical designation of another letter in the cluster of the core letter. The second number in each paired value represents the weight to be given the cluster letter as a substitute for the core letter.

The first pair is the core letter itself and the value to be given, when it is a match. For example, for the letter "a", the first pair "{0, 8} represents letter "0" (i.e. "a") and its match weight of 8. Continuing with the letter "a", the next pair is for the cluster letter "e", and its match weight of 4. the next two cluster letters, "o" (letter 14), and "s" (letter 18), have match weights of 2. The fourth cluster letter, "i", has a match weight of 4. For the letter "a", the letters "e" and "i" are more often by mistake than the letters "o" and "s".

The cluster table values associated with each letter represent letters that may have the same sound as a letter (i.e. "k" and hard "c", "s" and "c") or that are near to each other on a standard "qwerty" keyboard, and are therefore likely to be mis-stroke. The following cluster table is given by way of example only. Other cluster tables may be used without departing from the scope or spirit of the present invention.

```
cluster table: {
  a:  {{0,8}, {4,4}, {14,2}, {18,2}, {8,4}, {0,0}},
  b:  {{1,8}, {21,2}, {13,2}, {3,2}, {0,0}},
  c:  {{2,8}, {18,4}, {10,4}, {23,2}, {21,2}, {25,2}, {0,0}},
  d:  {{3,8}, {18,2}, {5,2}, {1,2}, {0,0}},
  e:  {{4,6}, {0,3}, {8,3}, {14,2}, {22,2}, {17,2}, {20,2}, {0,0}},
  f:  {{5,8}, {21,4}, {6,2}, {3,2}, {15,4}, {7,4}, {0,0}},
  g:  {{6,8}, {9,4}, {5,2}, {7,2}, {0,0}},
  h:  {{7,8}, {5,4}, {6,2}, {9,2}, {0,0}},
  i:  {{8,8}, {24,4}, {4,3}, {14,2}, {20,2}, {0,4}, {0,0}},
  j:  {{9,8}, {6,4}, {10,2}, {7,2}, {0,0}},
  k:  {{10,8}, {2,4}, {23,4}, {16,4}, {9,2}, {11,2}, {0,0}},
  l:  {{11,8}, {17,2}, {10,2}, {0,0}},
  m:  {{12,8}, {13,4}, {0,0}},
  n:  {{13,8}, {12,2}, {1,2}, {0,0}},
  o:  {{14,8}, {20,2}, {4,3}, {0,2}, {8,3}, {15,2}, {0,0}},
  p:  {{15,8}, {5,4}, {14,2}, {0,0}},
q:  {{16,8}, {10,4}, {22,2}, {0,0}},
r:  {{17,8}, {11,2}, {4,2}, {19,2}, {0,0}},
s:  {{18,8}, {2,4}, {25,4}, {23,4}, {0,2}, {3,2}, {0,0}},
t:  {{19,8}, {17,2}, {24,2}, {0,0}},
u:  {{20,8}, {14,2}, {8,2}, {4,2}, {22,4}, {0,0}},
v:  {{21,8}, {22,4}, {5,4}, {1,2}, {2,2}, {0,0}},
w:  {{22,8}, {21,4}, {16,2}, {4,2}, {20,4}, {0,0}},
x:  {{23,8}, {10,4}, {18,4}, {25,2}, {2,2}, {0,0}},
y:  {{24,8}, {20,2}, {19,2}, {8,4}, {0,0}},
z:  {{25,8}, {18,4}, {23,2}, {2,2}, {0,0}},
  (other character's cluster)
    {{26,8}, {0,0}},
    {{27,8}, {0,0}},
    {{28,8}, {0,0}},
    {{29,8}, {0,0}},
    {{30,8}, {0,0}},
    {{31,8}, {0,0}}
```

The use of a cluster table is optional in the present invention.

PROJECTION WITH POSITION WEIGHTS

In addition to, or instead of, the use of weights in clusters, weights, w, can be assigned to each position in the normalized images. When a symbol is projected in the image, the distribution value and cluster weight can be multiplied with the weight associated with the symbol's position. Note that the weights are assigned to the normalized image instead of the original string.

It is often the case that words are misspelled at the beginning rather than in the middle or end. The position table can be used to indicate that the first two positions of a word have twice the weight compared with others. Thus, the first two characters in the string will have more significant impact on the similarity comparison. So, if the beginnings of two words are the same, they are more likely to be the same word.

When using position weights with cluster tables, the projection becomes:

$$c_{n\,M(s_i)+|D|+j} = d_j{}^{*}\ w_{M(s_i)}{}^{*}\ u_{s_i\,n}$$

$$c_{n\,M(s_i)+|D|-j} = d_j{}^* \; w_{M(s_i)}{}^* \; u_{s_i\,n} \qquad (j=0,1,2,\ldots,|D|)$$

When using positionweights without clusters, the projection becomes:

$$c_{n\,M(s_i)+|D|+j} = d_j{}^* \; w_{M(s_i)}$$

$$c_{n\,M(s_i)+|D|-j} = d_j{}^* \; w_{M(s_i)} \qquad (j=0,1,2,\ldots,|D|)$$

The following code may be used to generate projections from a given symbol string:

```
 * NAME
 *    zfmprojs - Projection Matching:  generate projections
 * DESCRIPTION
 *    generate projections from the string given and touch
 *    characters reached
 */

static eword
zfmprojs(pe_p, str, slen, projs)
reg0 zfmpenv *pe_p;
text    *str;
eword   slen;
ub2     *projs;
{
    reg6 eword    pp;        /* count the positions in projection */
    reg1 ub2      *prjptrl;  /* pointers to go thru a proj */
    reg7 ub2      *prjptrr;
    reg3 ub2      *dptr;     /* pointer to go thru dist[] */
    reg2 eword    ss;        /* score for a position */
    reg8 zfmpclup clstptr;   /* pointer to go thru a cluster */
    reg3 text     ch;        /* a char in the cluster */
    reg9 text     core;      /* core char */
    reg14 eword   score;     /* score for the char */
    reg10 eword   cc;        /* count the chars in string */
    reg11 eword   sum;       /* total score */
    eword x0;                /* beginning of a distribution */
    /* following variables are copied from zfmpenv */

    reg13 eword size;         /* size of the char set */
    reg15 eword neighbors;    /* neighbors */
    reg12 ub2   *dist;        /* distribution */
    eword       closure;      /* size of the projection */
    eword       npos;         /* number of positions */
    ub2         *poswts;      /* pointed to the weight table */

    /* get info from the ZFMP structure */

    size      = pe_p->pe_size;
    neighbors = pe_p->pe_neighbors;
    closure   = pe_p->pe_closure;
    npos      = pe_p->pe_npos;
    dist      = pe_p->pe_dist;
    poswts    = pe_p->pe_poswts;

    /* initialize work areas */

    for (prjptrl = projs, pp = size * closure; pp; --pp, ++prjptrl)
    {
        *prjptrl = (ub2)0;
    }

    sum = (eword)0;  /* sum is accumulated */

    /* for each char (as a core) in the string */

    for (cc = (eword)1, ++slen; cc < slen; ++cc, ++str)
    {
  core = *str;

        /* check the range of the core */

        if (core >= size)
        {
            continue;
        }
    /* locate the char in our projection */

    if ((!cc) || (slen == 1))
    {
        x0 = (eword)0;  /* so that divived-by-0 won't happen */
    }
    else
    {
        x0 = cc * npos / slen;
    }

    /* get a cluster, for each char in the cluster, do ... */

    for (clstptr = (zfmpclup)pe_p->pe_clusters[core];
          clstptr->cl_sc;
          ++clstptr)
    {
   ch = clstptr->cl_ch;

        /* get the score and mutiply the weigth */

        score = (eword)clstptr->cl_sc * poswts[x0];

        /* The char is touched.  First compute the
           points at the peak, than set prjptrl and
           prjptrr at the left and the right of
           the peak, respectively. */

        prjptrl = projs + ch * closure + x0 + neighbors;
        sum += *prjptrl = (ub2)(score * dist[0]);
        prjptrr = (prjptrl--) + 1;

        /* Priptrl and prjptrr are moving toward left
           and right, away from the peak.  The position
           they point to have the same score, so that
           ss is only calculated once. */

        for (pp = neighbors, dptr = dist + 1;
                         pp;
                         --pp, --prjptrl, ++prjptrr, ++dptr)
                 {
                         ss = score * (*dptr);  /* compute a score */

                         /* I am not sure whether to accumulate
                            points or to keep the highest one */
#ifndef ZFMPACCUMULATE
                         if (ss > *prjptrl)
                         {
                             sum += ss - *prjptrl;
                             *prjptrl = (ub2)ss;
                         }
                         if (ss > *prjptrr)
                         {
                             sum += ss - *prjptrr;
                             *prjptrr = (ub2)ss;
                         }
#else
                         sum += ss + ss;
                         *prjptrl += (ub2)ss;
                         *prjptrr += (ub2)ss;
#endif
                 }
             }
         }

         return (sum);
     }
```

PROJECTION MATCHING

After a projection is generated, whether it be a simple projection, a projection with clusters, a projection with position weights, or a projection with both clusters and position weights, a comparison of the model projection and the query projection is made to determine the closeness of the match. The comparison is accomplished using a similarity function.

A projection is a series of closures concatenated together. The similarity function Θ of the preferred embodiment of the present invention is defined as follows:

$$\Theta(P_i, P_2) = 1 - \frac{\sum | p_{1i} - p_{2i} |}{\sum p_{1i} + \sum p_{2i}}$$

where P_1 and P_2 are two projections to be compared. When two projections are identical, or two original strings are identical, the similarity is 1. The lowest possible Θ is 0.

Figure 2 is a flow diagram illustrating the preferred embodiment of the present invention. At step 201, do zfmpopen() is performed. zfmpopen opens an environment in which other functions operate. It returns a handle to the open environment and this handle is kept and passed to other functions to refer to the same environment. poswts and dist are two 0-terminated integer arrays. They are used to adjust the behavior of the comparison mechanism. For example, the following setting:

```
    int poswts[] = { 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1,1, 1, 1, 1, 0 };
    int dist[] = { 21, 20, 18, 15, 10, 6, 4, 2, 0 };
```

gives more priority on the beginning of a string and compensates models that have their characters matched to nearby positions in the query. Usually poswts is longer than most of expected strings. The longer dist the more compensation is give on matched characters at different positions. But dist is not longer than poswts in the preferred embodiment of the present invention. An extreme case is:

```
    int poswts[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 };
    int dist[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 };
```

which sets the string matching to a letter frequency comparison. The parameter clusters is an array of clusters, each cluster corresponds to a character in the character set as the core. Code for implementing zfmpopen() is illustrated in Appendix A.

At step 202, zfmpquery() is executed on a query 203. The query is processed, a projection is calculated by calling zfmpprojs() at step 208, and the result is stored in memory. zfmpquery() sets a new query for an environment. Once a query is set, all comparisons made in the environment are based on this query. pe_h indicates the environment, query is the query string, and qlen is the length of query. Code for implementing zfmpquery() is illustrated in Appendix B.

At decision block 204, the argument "Get model?" is made. If the argument is true, the system proceeds to step 205. If the argument is false, their are no more models, and the system proceeds to step 206, where the best matches are displayed.

At step 205, zfmpmodel() is executed. A model is processed by calling zfmpprojs() at step 208. A projection is returned and compared to the projection of the query. Code for implementing zfmpmodel is illustrated in Appendix C. At step 207, the similarity value for each model as compared to the query is provided.

At step 208, zfmprojs() is used to normalize the input sequence and calculate and return its projection. zfmprojs() may use cluster tables and/or position weights as desired.

<u>EXAMPLE 1</u>

As noted, the present invention can be implemented without using cluster tables or position weight tables. The invention can be implemented using one or both of the cluster table or position weight tables if desired. In this example the query is "communicate" and the model is "comunicate" and no cluster table or position weight table is used.

```
Query: Communicate
Model: Comunicate
degree(maximum is 17) 16
similarity: 94.117645%:

query projection(a):   0 0 0 0 0 0 0 4 10 14 17 14 10 4 0 0
model projection(a):   0 0 0 0 0 0 0 4 10 14 17 14 10 4 0 0

query projection(c):   4 10 14 17 14 10 4 10 14 17 14 10 4 0 0 0
model projection(c):   4 10 14 17 14 10 4 10 14 17 14 10 4 0 0 0

query projection(e):   0 0 0 0 0 0 0 0 0 4 10 14 17 14 10 4
model projection(e):   0 0 0 0 0 0 0 0 0 4 10 14 17 14 10 4
```

```
query projection(i):   0 0 0 0 0 4 10 14 17 14 10 4 0 0 0 0
model projection(i):   0 0 0 0 0 4 10 14 17 14 10 4 0 0 0 0

query projection(m):   0 0 4 10 14 17 17 14 10 4 0 0 0 0 0 0
model projection(m):   0 0 4 10 14 17 14 10 4 0 0 0 0 0 0 0

query projection(n):   0 0 0 0 0 4 10 14 17 14 10 4 0 0 0 0
model projection(n):   0 0 0 0 4 10 14 17 14 10 4 0 0 0 0 0

query projection(o):   0 4 10 14 17 14 10 4 0 0 0 0 0 0 0 0
model projection(o):   0 4 10 14 17 14 10 4 0 0 0 0 0 0 0 0

query projection(t):   0 0 0 0 0 0 0 0 4 10 14 17 14 10 4 0
model projection(t):   0 0 0 0 0 0 0 0 4 10 14 17 14 10 4 0

query projection(u):   0 0 0 0 4 10 14 17 14 10 4 0 0 0 0 0
model projection(u):   0 0 0 4 10 14 17 14 10 4 0 0 0 0 0 0
```

Note that because cluster tables are not used, only the letters of the model (namely, a, c, e, i, m, n, o, t, and u), are used in the comparison. There are two peaks for the letter "c" because it is the first letter and the eighth letter in "communicate". There are two peaks for "m" in the query"communicate" but only one for the misspelled model "comunicate".

<u>EXAMPLE 2</u>

Example two illustrates a situation where a cluster table is used but the position weight table is not used. In this example, distribution table number 2 is used.

```
query: communicate
model: comunicate

degree(maximum is 136) 131
similarity: 96.323532%:

query projection(a):   0 8 20 28 34 28 40 56 80 112 136 112 51 42 30 12
model projection(a):   0 8 20 28 34 28 40 56 80 112 136 112 51 42 30 12
```

query projection(b):	0 0 0 0 0 8 20 28 34 28 20 8 0 0 0 0
model projection(b):	0 0 0 0 8 20 28 34 28 20 8 0 0 0 0 0
query projection(c):	32 80 112 136 112 80 32 80 112 136 112 80 32 0 0 0
model projection(c):	32 80 112 136 112 80 32 80 112 136 112 80 32 0 0 0
query projection(e):	0 12 30 42 51 42 30 42 51 56 68 84 102 84 60 24
model projection(e):	0 12 30 42 51 42 34 42 51 56 68 84 102 84 60 24
query projection(i):	0 12 30 42 51 42 80 112 136 112 68 56 51 42 30 12
model projection(i):	0 12 30 42 51 42 80 112 136 112 68 56 51 42 30 12
query projection(k):	16 40 56 68 56 40 16 40 56 68 56 40 16 0 0 0
model projection(k):	16 40 56 68 56 40 16 40 56 68 56 40 16 0 0 0
query projection(m):	0 0 32 80 112 136 136 112 34 32 20 8 0 0 0 0
model projection(m):	0 0 32 80 112 136 112 34 32 20 8 0 0 0 0 0
query projection(n):	0 0 16 40 56 68 80 112 136 112 80 32 0 0 0 0
model projection(n):	0 0 16 40 56 80 112 136 112 80 32 0 0 0 0 0
query projection(o):	0 32 80 112 136 112 80 34 34 28 34 28 34 28 20 8
model projection(o):	0 32 80 112 136 112 34 32 34 28 34 28 34 28 20 8
query projection(p):	0 8 20 28 34 28 20 8 0 0 0 0 0 0 0 0
model projection(p):	0 8 20 28 34 28 20 8 0 0 0 0 0 0 0 0
query projection(r):	0 0 0 0 0 0 0 0 8 20 28 34 34 28 20 8
model projection(r):	0 0 0 0 0 0 0 0 8 20 28 34 34 28 20 8
query projection(s):	16 40 56 68 56 40 16 40 56 68 34 40 20 8 0 0
model projection(s):	16 40 56 68 56 40 16 40 56 68 34 40 20 8 0 0
query projection(t):	0 0 0 0 0 0 0 0 32 80 112 136 112 80 32 0
model projection(t):	0 0 0 0 0 0 0 0 32 80 112 136 112 80 32 0
query projection(u):	0 8 20 28 34 80 112 136 34 80 32 28 34 28 20 8
model projection(u):	0 8 20 32 80 112 136 112 34 32 20 28 34 28 20 8

```
query projection(v):  8 20 28 34 28 20  8 20 28 34 28 20  8  0  0  0
model projection(v):  8 20 28 34 28 20  8 20 28 34 28 20  8  0  0  0

query projection(w):  0  0  0  0 16 40 56 68 56 40 20 28 34 28 20  8
model projection(w):  0  0  0 16 40 56 68 56 40 16 20 28 34 28 20  8

query projection(x):  8 20 28 34 28 20  8 20 28 34 28 20  8  0  0  0
model projection(x):  8 20 28 34 28 20  8 20 28 34 28 20  8  0  0  0

query projection(y):  0  0  0  0  0 16 40 56 68 56 40 34 28 20  8  0
model projection(y):  0  0  0  0  0 16 40 56 68 56 40 34 28 20  8  0

query projection(z):  8 20 28 34 28 20  8 20 28 34 28 20  8  0  0  0
model projection(z):  8 20 28 34 28 20  8 20 28 34 28 20  8  0  0  0
```

In this example, additional letters are tested because the cluster table is chosen. For example, the letter "a" is a core letter for the letters "e", "o", "s", and "i". The letter "c" is a core letter for the letters "s", "k", "x", "v", and "z". Therefore, the additional letters "b", "k", "p", "r", "s", "v", "w", "x", "y", and "z" are analyzed in addition to the letters in "communicate".

Referring to the results above, there are two peaks for the letter "k", corresponding to the position of the letter "c" in "communicate". This is because "k" is in the cluster of the letter "c". There are also two peaks for each of the letters "s", "v", "x", and "z", all cluster letters of the letter "c". The peaks for these letters are smaller than that for the letter "k" because their match weight is lower.

<u>EXAMPLE 3</u>

This example uses the position weight table, but not the cluster table. The position weight table is given by: { 2, 2, 1, 1, 1, 1, 1, 1, 1, 1 }. This means that the first two characters are given twice the weight as the remaining characters. This is because most spelling mistakes are made at the beginning of a word as opposed to the middle or end of a word. Distribution table number 2 is used.

query: communicate
model: comunicate

```
degree(maximum is 34) 32
similarity: 94.117645%:

query projection(a):  0 0 0 0 0 0 0 4 10 14 17 14 10 4 0 0
model projection(a):  0 0 0 0 0 0 0 4 10 14 17 14 10 4 0 0

query projection(c):  8 20 28 34 28 20 8 10 14 17 14 10 4 0 0 0
model projection(c):  8 20 28 34 28 20 8 10 14 17 14 10 4 0 0 0

query projection(e):  0 0 0 0 0 0 0 0 0 4 10 14 17 14 10 4
model projection(e):  0 0 0 0 0 0 0 0 0 4 10 14 17 14 10 4

query projection(i):  0 0 0 0 0 4 10 14 17 14 10 4 0 0 0 0
model projection(i):  0 0 0 0 0 4 10 14 17 14 10 4 0 0 0 0

query projection(m):  0 0 4 10 14 17 17 14 10 4 0 0 0 0 0 0
model projection(m):  0 0 4 10 14 17 14 10 4 0 0 0 0 0 0 0

query projection(n):  0 0 0 0 0 4 10 14 17 14 10 4 0 0 0 0
model projection(n):  0 0 0 0 4 10 14 17 14 10 4 0 0 0 0 0

query projection(o):  0 8 20 28 34 28 20 8 0 0 0 0 0 0 0 0
model projection(o):  0 8 20 28 34 28 20 8 0 0 0 0 0 0 0 0

query projection(t):  0 0 0 0 0 0 0 0 4 10 14 17 14 10 4 0
model projection(t):  0 0 0 0 0 0 0 0 4 10 14 17 14 10 4 0

query projection(u):  0 0 0 0 4 10 14 17 14 10 4 0 0 0 0 0
model projection(u):  0 0 0 4 10 14 17 14 10 4 0 0 0 0 0 0
```

The influence of the position weight table is seen in that the peaks for the first two letters, namely "c" and "o", are twice that of the peaks for the remaining letters, (34-17). Also note that the second occurrence of the letter "c" has only a peak of 17, versus the peak of 34 of the first occurrence.

EXAMPLE 4

This example uses both a cluster table and a position weight table. The effect of the cluster table is shown by the additional cluster letters that are analyzed.

The effect of the position weight table is shown for the letter "c" and "o", where the peak values are twice as high as for the other letters, (272-136). In addition, the first peaks for the cluster letters for "c", ("s", "k", "x", "v", and "z"), are twice as high as the second peak, illustrating the effect of the position weight table. The peaks for cluster letters for "o", ("u", "e", "a", "i", and "p"), are higher due to the position weight table.

```
query: communicate
model: comunicate

degree(maximum is 272) 263
similarity: 96.691177%:

query projection(a):  0 16 40 56 68 56 40 56 80 112 136 112 51 42 30 12
model projection(a):  0 16 40 56 68 56 40 56 80 112 136 112 51 42 30 12

query projection(b):  0 0 0 0 0 8 20 28 34 28 20 8 0 0 0 0
model projection(b):  0 0 0 0 8 20 28 34 28 20 8 0 0 0 0 0

query projection(c):  64 160 224 272 224 160 64 80 112 136 112 80 32 0 0 0
model projection(c):  64 160 224 272 224 160 64 80 112 136 112 80 32 0 0 0

query projection(e):  0 24 60 84 102 84 60 42 51 56 68 84 102 84 60 24
model projection(e):  0 24 60 84 102 84 34 42 51 56 68 84 102 84 60 24

query projection(i):  0 24 60 84 102 84 80 112 136 112 68 56 51 42 30 12
model projection(i):  0 24 60 84 102 84 80 112 136 112 68 56 51 42 30 12

query projection(k):  32 80 112 136 112 80 32 40 56 68 56 40 16 0 0 0
model projection(k):  32 80 112 136 112 80 32 40 56 68 56 40 16 0 0 0

query projection(m):  0 0 32 80 112 136 136 112 34 32 20 8 0 0 0 0
model projection(m):  0 0 32 80 112 136 112 34 32 20 8 0 0 0 0 0

query projection(n):  0 0 16 40 56 68 80 112 136 112 80 32 0 0 0 0
model projection(n):  0 0 16 40 56 80 112 136 112 80 32 0 0 0 0 0

query projection(o):  0 64 160 224 272 224 160 34 34 28 34 28 34 28 20 8
model projection(o):  0 64 160 224 272 224 34 64 34 28 34 28 34 28 20 8
```

query projection(p):	0 16 40 56 68 56 40 16 0 0 0 0 0 0 0 0
model projection(p):	0 16 40 56 68 56 40 16 0 0 0 0 0 0 0 0
query projection(r):	0 0 0 0 0 0 0 0 8 20 28 34 34 28 20 8
model projection(r):	0 0 0 0 0 0 0 0 8 20 28 34 34 28 20 8
query projection(s):	32 80 112 136 112 80 32 40 56 68 34 40 20 8 0 0
model projection(s):	32 80 112 136 112 80 32 40 56 68 34 40 20 8 0 0
query projection(t):	0 0 0 0 0 0 0 0 32 80 112 136 112 80 32 0
model projection(t):	0 0 0 0 0 0 0 0 32 80 112 136 112 80 32 0
query projection(u):	0 16 40 56 68 80 112 136 34 80 32 28 34 28 20 8
model projection(u):	0 16 40 56 80 112 136 112 34 32 20 28 34 28 20 8
query projection(v):	16 40 56 68 56 40 16 20 28 34 28 20 8 0 0 0
model projection(v):	16 40 56 68 56 40 16 20 28 34 28 20 8 0 0 0
query projection(w):	0 0 0 0 16 40 56 68 56 40 20 28 34 28 20 8
model projection(w):	0 0 0 16 40 56 68 56 40 16 20 28 34 28 20 8
query projection(x):	16 40 56 68 56 40 16 20 28 34 28 20 8 0 0 0
model projection(x):	16 40 56 68 56 40 16 20 28 34 28 20 8 0 0 0
query projection(y):	0 0 0 0 0 16 40 56 68 56 40 34 28 20 8 0
model projection(y):	0 0 0 0 0 16 40 56 68 56 40 34 28 20 8 0
query projection(z):	16 40 56 68 56 40 16 20 28 34 28 20 8 0 0 0
model projection(z):	16 40 56 68 56 40 16 20 28 34 28 20 8 0 0 0

EXAMPLE 5

This example illustrates a comparison of "communicate" and "comunicate" without cluster table and without position weights, but using distribution table number 1.

query: communicate
model: comunicate

```
degree(maximum is 21) 20
similarity: 95.238098%:

query projection(a):  0 0 0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0
model projection(a):  0 0 0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0

query projection(c):  4 10 14 17 19 21 19 17 14 17 19 21 19 17 14 10 4 0 0 0
model projection(c):  4 10 14 17 19 21 19 17 14 17 19 21 19 17 14 10 4 0 0 0

query projection(e):  0 0 0 0 0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4
model projection(e):  0 0 0 0 0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4

query projection(i):  0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0
model projection(i):  0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0

query projection(m):  0 0 4 10 14 17 19 21 21 19 17 14 10 4 0 0 0 0 0 0
model projection(m):  0 0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0 0 0 0

query projection(n):  0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0
model projection(n):  0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0 0

query projection(o):  0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0 0 0 0 0
model projection(o):  0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0 0 0 0 0

query projection(t):  0 0 0 0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0
model projection(t):  0 0 0 0 0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0

query projection(u):  0 0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0 0
model projection(u):  0 0 0 4 10 14 17 19 21 19 17 14 10 4 0 0 0 0 0 0
```

EFFECTS OF OPTIONAL TABLES

If a word such as "communicate" is misspelled as "comunicate", generally we may say that these two words have 1 character different out of 11 characters. Thus, "comunicate" is 91% similar to "communicate". However, the actual similarity of the two words is higher. Using the present invention, with cluster tables, position weights and distribution table number 1, the similarity becomes approximately 97%.

Now compare the result of comparing "communicate" with "communikate" and "communigate". With cluster table above, a better similarity comes out when "communicate" is compared with "communikate" than compared with "communigate" (94.3% vs 91.7%). It means that "communikate" is more likely to be "communicate" than "communigate" is, since "k" and "c" sometimes may have same pronunciation, and "k" is in the cluster of "c". With the position weight table a better similarity (94.3% vs 92.2%) is achieved while comparing "communicate" with "communikate".

A block diagram of the preferred embodiment of the present invention is illustrated in Figure 3. A query string 301 is provided to normalizing block 302. Model vectors from model storage 313 are provided to normalizing block 302 on line 311. The normalizing block 302 normalizes the data string of S symbols into a normalized image of N symbols. The normalized image 303A of the query and the normalized image 303B of the model vector are provided to the projection generating block 304.

A first memory means 305 for storing a cluster table is switchably coupled to projection generating means 304 through switch 307. A second memory means 306 for storing a position weight table is switchably coupled to projection generating means 304 through switch 308. Switches 307 and 308 can be independently controlled so that the projection vector 309 generated by projection generating block 304 may optionally include the effect of the cluster table 305 and/or the position weight table 306.

The projection vector 309A of the normalized query 303A, and the projection vector 309B of the normalized model vector 303B, are provided to projection matching block 309. The projection matching block 310 generates a similarity value 312, representing the degree of similarity between the projection vector 309A of the query and the projection vector 309B of the model vector. The projection matching block 310 operates in accordance with the algorithm:

$$\Theta(P_1, P_2) = 1 - \frac{\sum | p_{1i} - p_{2i} |}{\sum p_1 + \sum p_{2i}}$$

where P_1 and P_2 are two projections to be compared. When two projections are identical, or two original strings are identical, the similarity is 1. The lowest possible Θ is 0.

The first, second, and third memory means of Figure 3 can be implemented as three address regions in a single memory. In addition, the apparatus of Figure 3 can be implemented on a processor as a plurality of processor executable instructions.

Thus, a method and apparatus for comparing data strings has been described.

APPENDIX A

```
 *     zfmpopen - Projection Matching:  open a ZFMP structure
 * DESCRIPTION
 *     allocate and initialize zfmpenv structure
 */


zfmpref *
zfmpopen(size, maxsim, poswts, dist, clusters)
reg6  eword    size;
reg12 eword    maxsim;
reg8  ub2      *poswts;
reg7  ub2      *dist;
reg13 zfmpclut *clusters;
{
    reg0  zfmpenv  *pe_p;  /* pointer to return */
    reg1  eword    i;

    /* following variables are calculated from the parameters */

    reg4   eword neighbors;
    reg5   eword closure;
    reg10  eword npos;

    /* We use array indexes instead of pointers, because we don't
       want to distroy dist and poswts.  The overhead is minor
       since zfmpopen is only called once for each session. */

    for (i = 0; dist[i]; ++i);  /* [sic] how many neighbors */
    neighbors = i - 1;

    for (i = 0; poswts[i]; ++i);  /* [sic] how many positions */
    closure = i + neighbors * 2;
    npos    = i;

#ifdef DEBUG
    printf("neighbors = %d, closure = %d, npos = %d\n",
      neighbors, closure, npos);
    printf("dist[]: ");
```

```
        for (i = 0; i < neighbors + 1; ++i) printf("%d ", dist[i]);
        printf("\nposwts[]: ");
        for (i = 0; i < npos; ++i) printf("%d ", poswts[i]);
        printf("\n");
#endif

        /* allocate ZFMP environment */

        if (!(pe_p = (zfmpenv *)malloc(sizeof(zfmpenv))))
        {
            return ((zfmpref *)0);
        }

        pe_p->pe_size       = size;
        pe_p->pe_maxsim     = maxsim;
        pe_p->pe_closure    = closure;
        pe_p->pe_neighbors  = neighbors;
        pe_p->pe_npos       = npos;
        pe_p->pe_dist       = dist;
        pe_p->pe_poswts     = poswts;
        pe_p->pe_clusters   = clusters;
        pe_p->pe_qprojs     = 0;
        pe_p->pe_mprojs     = 0;

        /* allocate memory */

        if (!(pe_p->pe_qprojs = (ub2 *)malloc(sizeof(ub2) * size * closure))
||
      !(pe_p->pe_mprojs = (ub2 *)malloc(sizeof(ub2) * size * closure)))
        {
      zfmpclose((zfmpref *)pe_p);
      return ((zfmpref *)0);
        }

        /* cast and return */

        return ((zfmpref *)pe_p);
}
```

APPENDIX B

```
 * NAME
 *     zfmpquery - Projection Matching:  set a query
 * DESCRIPTION
 *     set the query to the string given and generate its projections
 */

void
zfmpquery(pe_h, query, qlen)
reg0 zfmpref *pe_h;
reg1 text    *query;
reg2 eword   qlen;
{
    /* do projections */

    zfmp_c(pe_h)->pe_qsum = zfmprojs(zfmp_c(pe_h), query,
                                                   qlen,
                                                   zfmp_c(pe_h)-
>pe_qprojs);
#ifdef DEBUG
    {
        int i, j;
   int qsum;

   qsum = 0;

        for (i = 0; i < zfmp_c(pe_h)->pe_size; ++i)
        {
            printf("%c: ", i + 'a');
            for (j = 0; j < zfmp_c(pe_h)->pe_closure; ++j)
            {
                printf("%d ", zfmp_c(pe_h)->pe_qprojs[i][j]);
                                                  qsum += zfmp_c(pe_h)-
>pe_qprojs[i][j];
            }
            printf("\n");
        }
        printf("pe_qsum = %d, qsum = %d\n", zfmp_c(pe_h)->pe_qsum, qsum);
         }
    #endif
    }
```

APPENDIX C

```
 * NAME
 *     zfmpmodel - compute the similarity index
 * DESCRIPTION
 *     generate projections for the model and compare the projections
 *     to those of query
 */

eword
zfmpmodel(pe_h, model, mlen)
reg0  zfmpref *pe_h;
reg11 text    *model;
reg12 eword   mlen;
{
    reg6  eword   i;
    reg7  ub4     sigma;       /* total of projections */
    reg9  eword   delta;       /* difference between two prjections */
    reg8  ub2     *qprojs;     /* projections from zfmpenv.*/
    reg5  ub2     *mprojs;

    /* get pointers */

    qprojs   = zfmp_c(pe_h)->pe_qprojs;
    mprojs   = zfmp_c(pe_h)->pe_mprojs;

    /* do projections for the model and get the sigma */

    sigma = (ub4)zfmp_c(pe_h)->pe_qsum +
            zfmprojs(zfmp_c(pe_h), model,
                                        mlen,
                                        mprojs);

    /* calculate the difference */

    delta = (eword)0;

    for (i = zfmp_c(pe_h)->pe_size * zfmp_c(pe_h)->pe_closure;
    i;
```

```
            --i, ++qprojs, ++mprojs)
        {
            delta += *qprojs > *mprojs? (eword)*qprojs - *mprojs:
                                        (eword)*mprojs - *qprojs;
        }

        return ((eword)((sigma - delta) * (zfmp_c(pe_h)->pe_maxsim) /
    sigma));
    }
```

CLAIMS

1. A method of comparing a first string of symbols with a second string of symbols, said method comprising the steps of:

normalizing said first string to create a first normalized string;

generating a first projection from said first normalized string;

normalizing said second string to create a second normalized string;

generating a second projection from said second normalized string;

comparing said first projection and said second projection to determine a degree of similarity of said first and second projections.

2. The method of claim 1 wherein said steps of generating said first projection and said second projection include the use of cluster tables.

3. The method of claim 1 wherein said steps of generating said first projection and said second projection include the use of position weight tables.

4. The method of claim 1 wherein said steps of generating said first projection and said second projection include the use of cluster tables and position weight tables.

5. The method of claim 1 wherein said step of normalizing said first string comprises generating a medium of a symbol, M, in a normalized image by:

$$M(S_i) = i * |N| / |S|$$

where S_i is the i-th symbol in a string S, $|N|$ is the normalized size, and $|S|$ is the length of string S

6. The method of claim 1 wherein said projection of said first string is generated by projecting said first string onto its closure in a normal distribution by:

$$c_{s_i M(S_i)+|D|+j} = d$$

$$c_{s_i M(S_i)+|D|-j} = d_j \qquad (j = 0, 1, 2, \ldots, |D|)$$

where D is a distributing series, |D| is distribution size, d_j is the j-th item in distribution series D, and $c_{s_i k}$ is the k-th item in symbol S_i's closure.

7. The method of claim 2 wherein said step of generating said projection of said first string with the use of a cluster table is accomplished by:

$$c_{n M(S_i)+|D|+j} = d_j{}^{*}\ u_{S_i n}$$

$$c_{n M(S_i)+|D|-j} = d_j{}^{*}\ u_{S_i n} \qquad (j = 0, 1, 2, \ldots, |D|)$$

where D is a distributing series, |D| is distribution size, d_j is the j-th item in distribution series D, $c_{s_i k}$ is the k-th item in symbol S_i's closure, and $u_{s_i n}$ is weight of symbol n in the cluster whose core is S_i.

8. The method of claim 3 wherein said step of generating said projection of said first string with the use of position weight tables is accomplished by:

$$c_{nM(S_i)+|D|+j} = d_j{}^{*}\ w_{M(S_i)}$$

$$c_{nM(S_i)+|D|-j} = d_j{}^{*}\ w_{M(S_i)} \qquad (j=0, 1, 2, \ldots, |D|)$$

where D is a distributing series, |D| is distribution size, d_j is the j-th item in distribution series D, $c_{s_i k}$ is the k-th item in symbol S_i's closure and $w_{M(S_i)}$ is a weight on position $M(S_i)$.

9. The method of claim 4 wherein said step of generating said projection of said first string with the use of a cluster table and a weight table and is accomplished by:

$$c_{n M(S_i)+|D|+j} = d_j{}^{*}\ w_{M(S_i)}{}^{*}\ u_{S_i n}$$

$$c_{n\,M(S_i)+|D|-j} = d_j{}^{*}\ w_{M(S_i)}{}^{*}\ u_{S_i\,n} \qquad (j=0, 1, 2, \ldots, |D|)$$

where D is a distributing series, $|D|$ is distribution size, d_j is the j-th item in distribution series D, c_{s_ik} is the k-th item in symbol S_i's closure, u_{s_in} is a cluster weight, and w is a position weight.

10. Apparatus for comparing a first string of symbols with a second string of symbols comprising:

normalizing means for normalizing said first string to create a first normalized string and for normalizing said second string to create a second normalized string;

projection generating means coupled to said normalizing means for generating a first projection from said first normalized string and for generating a second projection from said second normalized string;

comparing means coupled to said projection generating means for comparing said first projection and said second projection to determine a degree of similarity of said first and second projections.

11. The apparatus of claim 10 wherein generating said first projection and said second projection is accomplished with the use of cluster tables.

12. The apparatus of claim 10 wherein generating said first projection and said second projection is accomplished with the use of position weight tables.

13. The apparatus of claim 10 wherein generating said first projection and said second projection is accomplished with the use of cluster tables and position weight tables.

14. The apparatus of claim 10 wherein normalizing said first string comprises generating a medium of a symbol, M, in a normalized image by:

$$M(S_i) = i * |N| / |S|$$

where S_i is the i-th symbol in a string S, $|N|$ is the normalized size, and $|S|$ is the length of string S

15. The apparatus of claim 10 wherein said projection of said first string is generated by projecting said first string onto its closure in a normal distribution by:

$$c_{s_i\, M(S_i)+|D|+j} = d$$

$$c_{s_i\, M(S_i)+|D|-j} = d_j \qquad (j = 0, 1, 2, ..., |D|)$$

where D is a distributing series, $|D|$ is distribution size, d_j is the j-th item in distribution series D, and c_{s_ik} is the k-th item in symbol S_i's closure.

16. The apparatus of claim 11 wherein generating said projection of said first string with the use of a cluster table is accomplished by:

$$c_{n\, M(S_i)+|D|+j} = d_j{}^{*}\ u_{S_i n}$$

$$c_{n\, M(S_i)+|D|-j} = d_j{}^{*}\ u_{S_i n} \qquad (j = 0, 1, 2, ..., |D|)$$

where D is a distributing series, $|D|$ is distribution size, d_j is the j-th item in distribution series D, c_{s_ik} is the k-th item in symbol S_i's closure, and u_{s_in} is a cluster weight.

17. The apparatus of claim 12 wherein generating said projection of said first string with the use of position weight tables is accomplished by:

$$c_{n\, M(S_i)+|D|+j} = d_j{}^{*}\ w_{M(S_i)}$$

$$c_{n\, M(S_i)+|D|-j} = d_j{}^{*}\ w_{M(S_i)} \qquad (j=0, 1, 2, ..., |D|)$$

where D is a distributing series, $|D|$ is distribution size, d_j is the j-th item in distribution series D, c_{s_ik} is the k-th item in symbol S_i's closure and w is a position weight.

18. The apparatus of claim 13 wherein generating said projection of said first string with the use of a cluster table and a weight table and is accomplished by:

$$c_{n\,M(s_i)+|D|+j} = d_j{}^* \; w_{M(s_i)}{}^* \; u_{s_i\,n}$$

$$c_{n\,M(s_i)+|D|-j} = d_j{}^* \; w_{M(s_i)}{}^* \; u_{s_i\,n} \qquad (j=0, 1, 2, \ldots, |D|)$$

where D is a distributing series, |D| is distribution size, d_j is the j-th item in distribution series D, c_{s_ik} is the k-th item in symbol S_i's closure, u_{s_in} is a cluster weight, and $w_{m(Si)}$ is a position weight.

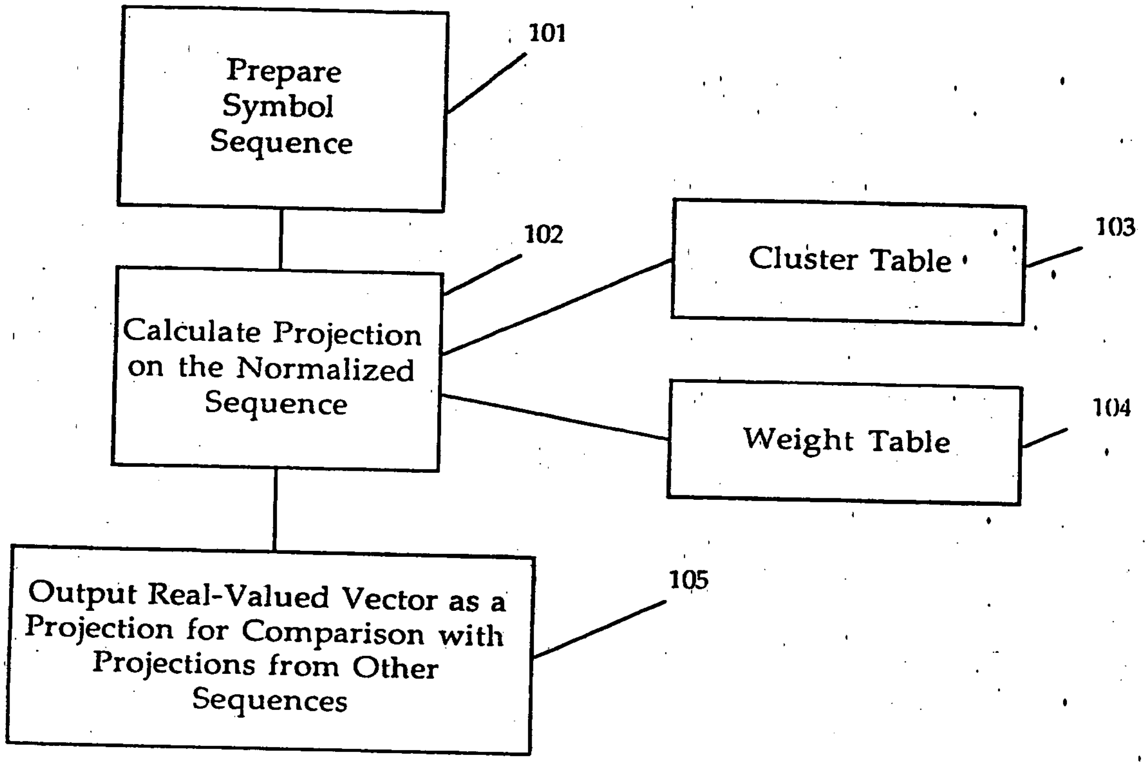

Prepare Symbol Sequence
101
Calculate Projection on the Normalized Sequence
102
Cluster Table
103
Weight Table
104
Output Real-Valued Vector as a Projection for Comparison with Projections from Other Sequences
105

Figure 1

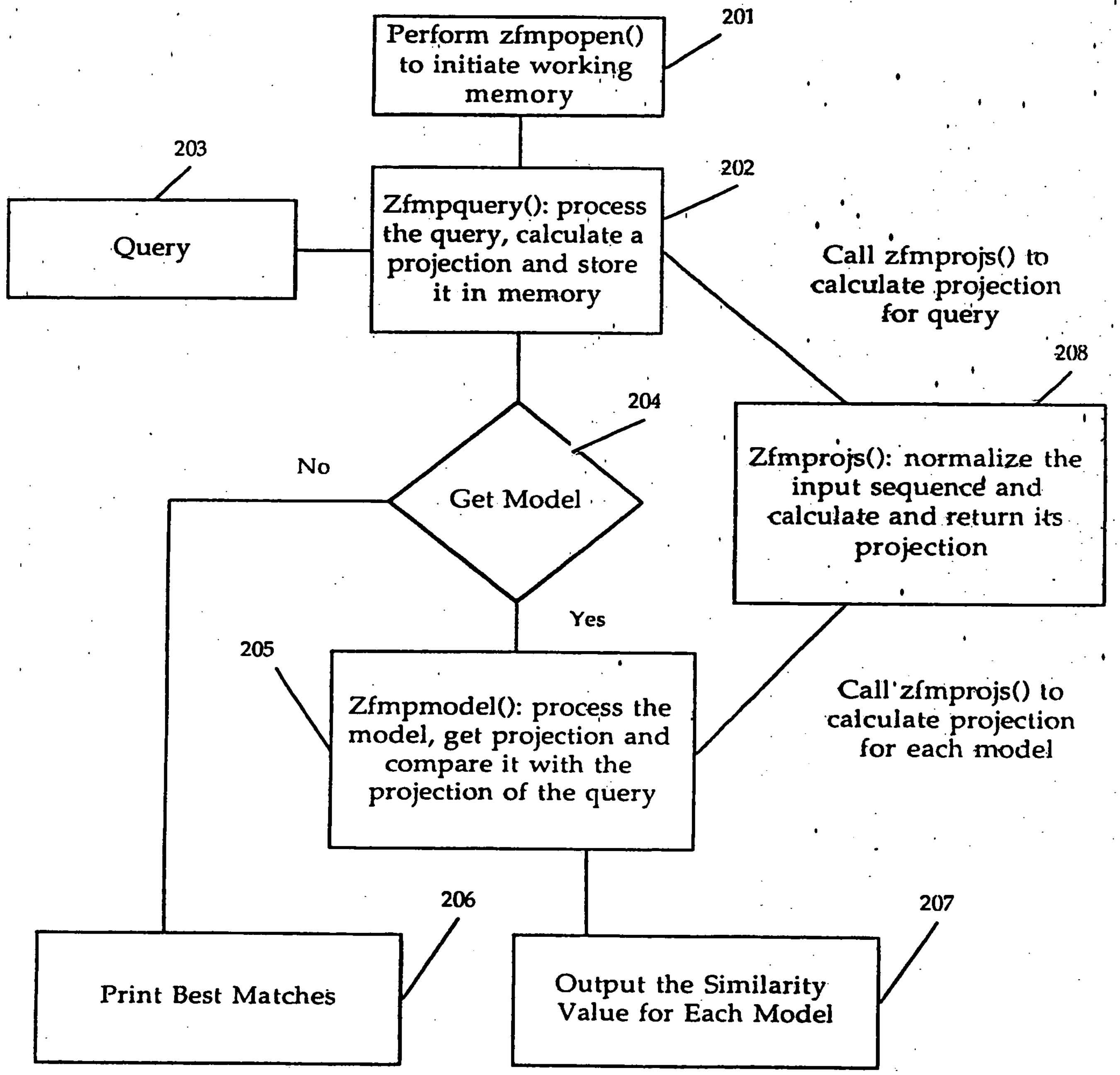

201
Perform zfmpopen() to initiate working memory
202
Zfmpquery(): process the query, calculate a projection and store it in memory
203
Query
Call zfmprojs() to calculate projection for query
208
Zfmprojs(): normalize the input sequence and calculate and return its projection
204
Get Model
No
Yes
205
Zfmpmodel(): process the model, get projection and compare it with the projection of the query
Call zfmprojs() to calculate projection for each model
206
Print Best Matches
207
Output the Similarity Value for Each Model

Figure 2

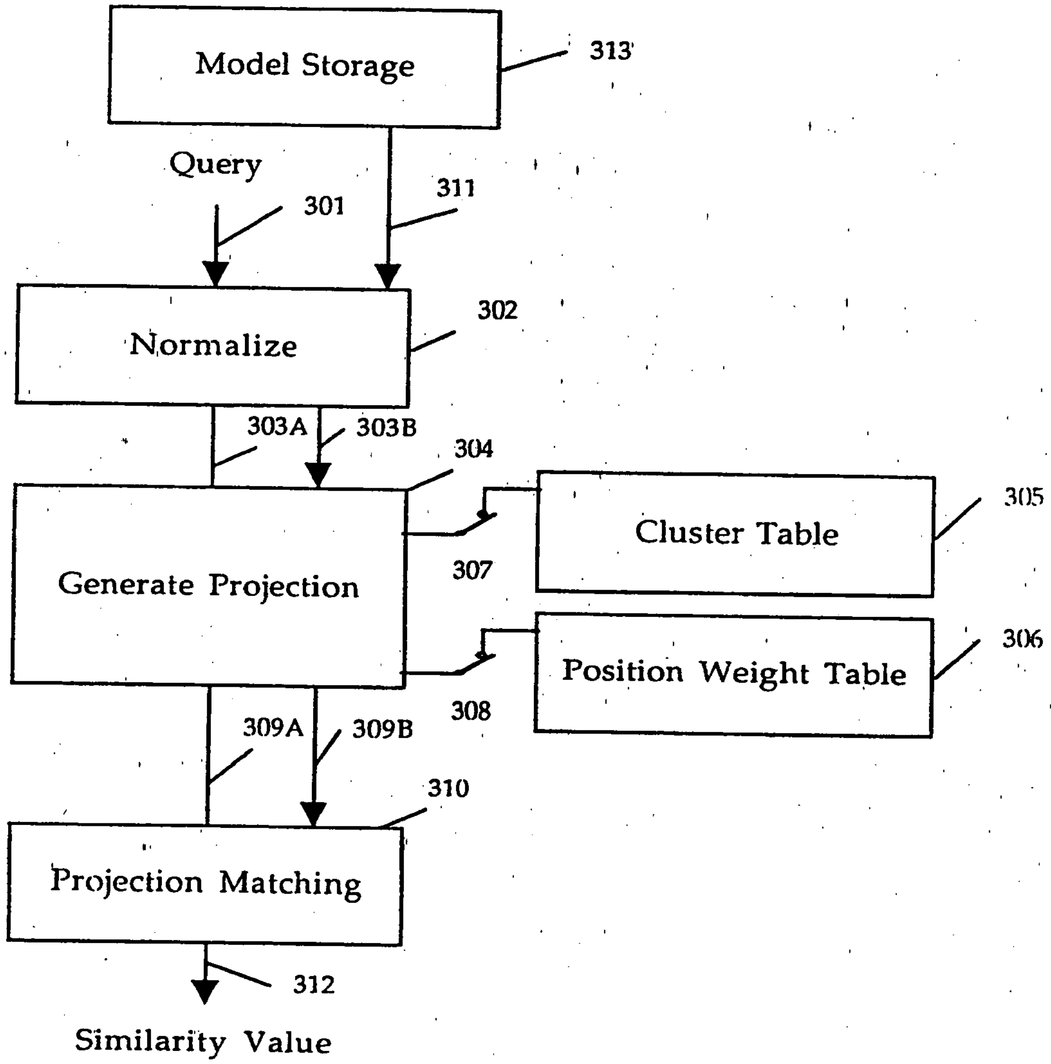

Model Storage
313
Query
301
311
Normalize
302
303A
303B
304
Cluster Table
305
Generate Projection
307
Position Weight Table
306
308
309A
309B
310
Projection Matching
312
Similarity Value

Figure 3

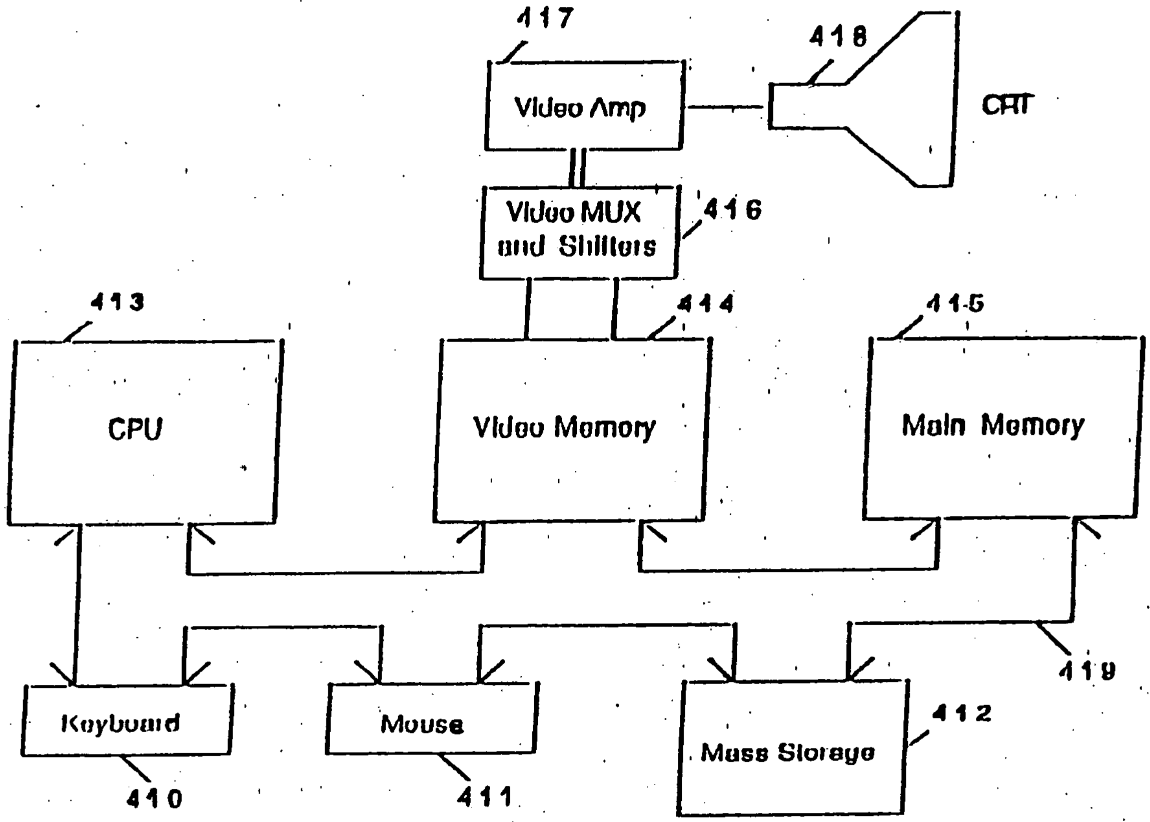

417
Video Amp
418
CRT
Video MUX and Shifters
416
413
CPU
414
Video Memory
415
Main Memory
419
Keyboard
410
Mouse
411
Mass Storage
412

FIGURE 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US93/02179

A. CLASSIFICATION OF SUBJECT MATTER

IPC(5) :G06K 9/62
US CL :382/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 382/11,27,28,38,41,42,40;364/715.09,715.11;395/2,51,54,934; 434/167

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

normalize project,projects,projection,projected,vecton, match,maletie,matched,matcher,matching,cluster # weight, sequence

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US,A, 3,803,553 (Nakano et al.) 09 April 1974 See column 5 and figures 1a and 1b.	1 and 10
A	US,A, 4,489,435 (Moshier et al.) 18 December 1984.	1-18
P,A,	US,A, 5,109,431 (Nishiya et al.) 28 April 1992.	1-18
A	US,A, 3,995,254 (Rosenbaum) 30 November 1976.	1-18
A	US,A, 5,033,097 (NAKAMURA) 16 JULY 1991	1-18

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be part of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 19 MAY 1993	Date of mailing of the international search report 09 JUL 1993
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer [illegible] MICHAEL CAMMARATA
Facsimile No. NOT APPLICABLE	Telephone No. (703) 305-4784